**[¿Qué son las propiedades estáticas?](https://www.youtube.com/watch?v=f5yazbsXRcs)**

Son variables que pertenecen a la clase en la que se declaran en lugar de a cualquier instancia de esa clase.

Esto significa que el valor de una propiedad estática es compartido entre todas las instacias de una Clase.

**¿Cómo se declaran las propiedades estáticas?**

Con la palabra clave “**static”** y son accesible sin necesidad de crear una instancia de clase. Por lo general, se utilizan para guardar datos que deben ser consistentes entre todas las instancias de una clase o para funciones que pueden operar sin requerí un objeto instanciado.

1. Desarrolle el siguiente ejemplo de la ilustración 1., anexe el enlace del repositorio.

|  |
| --- |
|  |

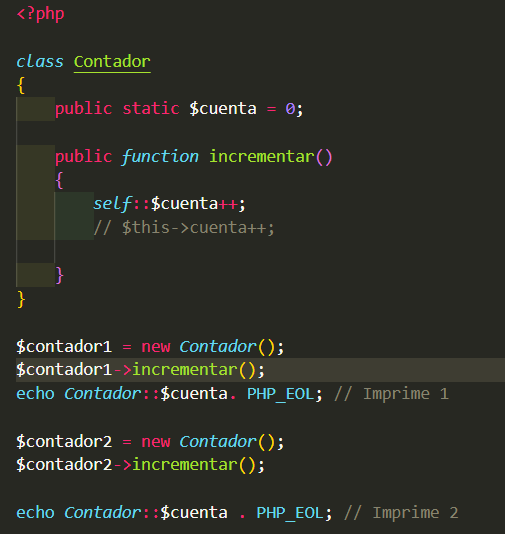
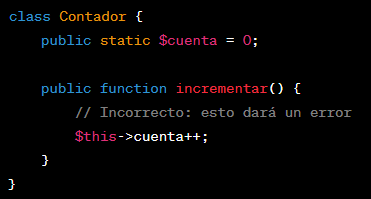
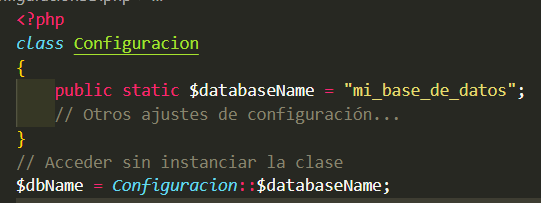
En el ejemplo, $cuenta es una propiedad estática de la clase Contador. Cuando se llama al método **incrementar()**, la propiedad estática $cuenta se incrementa. Aunque estamos utilizando dos instancias diferentes de Contador, la propiedad **$cuenta** es la misma para ambas, ya que es estática y pertenece a la clase misma.  
**Como acceder a propiedades estáticas:** Para acceder a una propiedad estática dentro de la clase, puedes usar la palabra clave self seguida del operador de resolución de ámbito **“**::**”** y el nombre de la propiedad. Desde fuera de la clase, puedes acceder a la propiedad estática directamente utilizando el nombre de la clase, seguido del operador de resolución de ámbito y el nombre de la propiedad, como **Contador::$cuenta**.

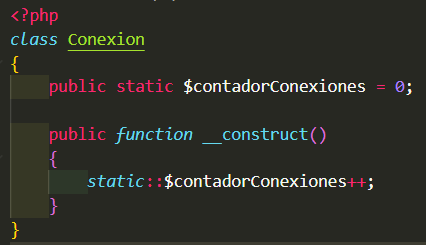
Ilustración 1

No se puede utilizar $this para acceder a propiedades estáticas. La variable $this es una referencia al objeto actual, no a la clase actual, y las propiedades estáticas pertenecen a la clase en lugar de a cualquier objeto en particular.Las propiedades estáticas se acceden utilizando el nombre de la clase o las palabras clave self:: o static:: cuando se accede desde dentro de métodos de la clase.

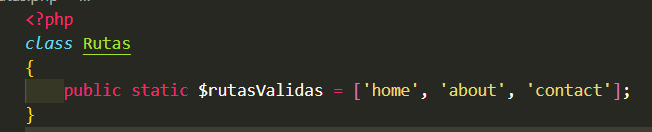


Las propiedades estáticas suelen usarse en varios tipos de proyectos y situaciones.

**Valores de configuración o ajustes globales:** Para almacenar información de configuración que necesita ser accesible en todas partes de una aplicación sin tener que crear una instancia de clase.



**Contadores y estadísticas:** Si se necesita llevar cuentas de cuantas instancias de una clase se han creado o cuántas veces se ha realizado una acción en particular, puedes usar una propiedad estática.

**Definición de contantes como:**

Las propiedades estáticas a veces se utilizaban como una forma de definir constantes que podrían contener arrays.

**¿Qué son los métodos estáticos?**

Es un método que pertenece a la clase en la que se define en lugar de a cualquier objeto que pueda ser una instancia de esa clase. Esto significa que puedes llamar a un método estático sin crear una instancia de la clase.

Los métodos estáticos se declaran con la palabra clave static y se acceden utilizando el operador de resolución de ámbito (::), seguido del nombre del método.

1. Implemente el siguiente código Ilustración 2, anexe el enlace del repositorio

|  |
| --- |
|  |

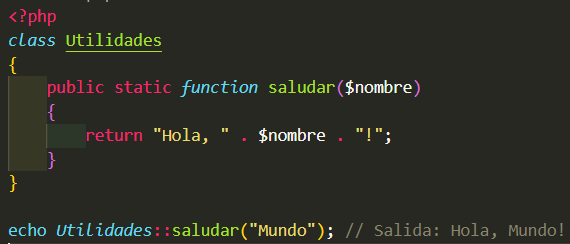
En este ejemplo, el método **saludar** es estático, lo que significa que puedes llamarlo directamente de la clase **Utilidades** sin necesidad de instanciar la clase primero

Ilustración 2

Los métodos estáticos son útiles en varias situaciones, por ejemplo:

* Cuando un método realiza una operación que no depende del estado de una instancia de la clase (no utiliza $this).
* Para proporcionar utilidades o funciones de ayuda que son relevantes para la clase pero que no necesitan acceder a los datos de instancia.
* Cuando mantienes un contador o un valor de configuración que es común para todas las instancias de la clase.
* En patrones de diseño como Singleton, Factory, o para implementar métodos de arranque o inicialización que pueden configurar ciertos aspectos antes de que se cree cualquier instancia.

Es importante recordar que, dado que los métodos estáticos no tienen acceso al pseudo-variable **$this**, no pueden acceder a las propiedades no estáticas de la clase directamente**.**

*El uso excesivo de métodos estáticos puede llevar a un diseño de software menos flexible y más difícil de probar, debido a la naturaleza global del estado estático y las dependencias que pueden crear entre clases. Por lo tanto, aunque son muy útiles, los métodos estáticos deben usarse con consideración.*

**Late Static Binding (LSB)**Permite referirse a la clase llamada en un contexto de herencia, más que la clase en la que se ejecuta el método.

[**¿Cómo funciona Late Static Binding?**](https://www.youtube.com/shorts/0F4v_L-UDfk)

PHP utiliza la palabra clave **static** en lugar de **self** para referirse a la clase llamada. Mientras que self se refiere a la clase donde el método se define, **static** se refiere a la clase que está siendo invocada en el momento de la ejecución.

1. Implemente el siguiente código de la Ilustración 3, anexe el enlace del repositorio.

|  |
| --- |
|  |

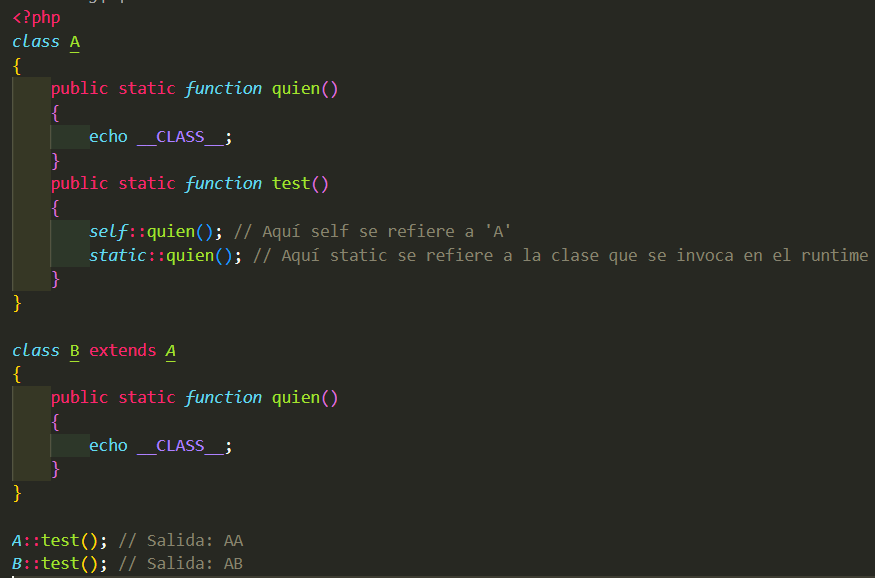


Ilustración 3

En el ejemplo anterior, cuando se llama a **B::test()**, la salida es AB porque **static::quien()** en **A::test()** se resuelve como **B::quien()** en tiempo de ejecución, ya que B es la clase que invoca el método.

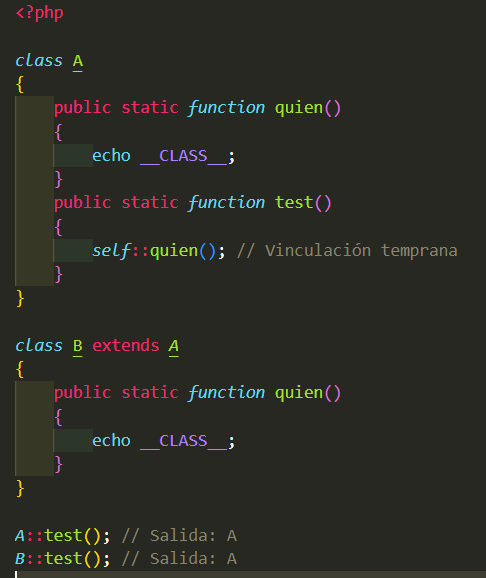
**Ventajas del Late Static Binding:**

* Flexibilidad en herencia: Permite que los métodos heredados puedan ser sobrescritos de manera más flexible.
* Reutilización de código: Permite reutilizar métodos de la clase padre y modificar su comportamiento en la clase hija.
* Polimorfismo estático: Facilita una forma de polimorfismo estático en el que el mismo método estático puede tener diferentes comportamientos dependiendo de la clase que lo invoca.

El LSB es una característica avanzada que proporciona un control más preciso y flexible sobre el comportamiento de los métodos estáticos en la herencia de clases. Es especialmente útil en patrones de diseño complejos y en la construcción de frameworks y bibliotecas que requieren una estructura de herencia sofisticada.

**Early Binding (Vinculación temprana)**

Mecanismo por el cual las referencias a métodos y propiedades se resuelven en tiempo de compilación, y no en tiempo de ejecución como es en el caso de “Late Binding”.



En este ejemplo, **self::quien()** siempre hará referencia al método definido en la clase A, sin importar si el método **test**() es llamado desde la clase **A** o **B**. Esto es porque la referencia se resuelve cuando el código es compilado y self está vinculado a la clase **A**.

**Cómo trabajar con Early Binding en PHP:** Usar “**self**” para referirse a la clase actual donde se define el método.

* Usar el nombre de la clase explícitamente para referirse a métodos o propiedades estáticas.
* Entender que cualquier referencia a métodos o propiedades estáticas se resolverá en tiempo de compilación y no será afectada por el tipo de la instancia en tiempo de ejecución.

**Desventajas del Early Binding:**

* ***Falta de flexibilidad***: No puedes sobrescribir el comportamiento de métodos estáticos heredados de la manera que lo harías con el Late Static Binding.
* ***Limitaciones en herencia:*** Cuando se desea que una clase heredada altere el comportamiento de un método estático, necesitas sobrescribir el método por completo en lugar de extender su funcionalidad.

**¿Cómo acceder a la propiedad estática en una clase hija?**

Dependiendo de la visibilidad de la propiedad (si es “public”,”protected” o “privare”), podrás acceder a ella directamente o a través de encapsulamiento. En muchos lenguajes de programación, incluido PHP, la vinculación temprana se da por defecto en la mayoría de los casos.

Cuando se utiliza la palabra clave **self** o se hace referencia directamente a una clase por su nombre, se está empleando vinculación temprana.

Esto significa se determinará qué método o propiedad está siendo referenciado en el momento en que el código es compilado, y esa referencia no cambiará en tiempo de ejecución.

1. Implemente el siguiente código de la Ilustración 4, anexe el enlace del repositorio.

|  |
| --- |
|  |

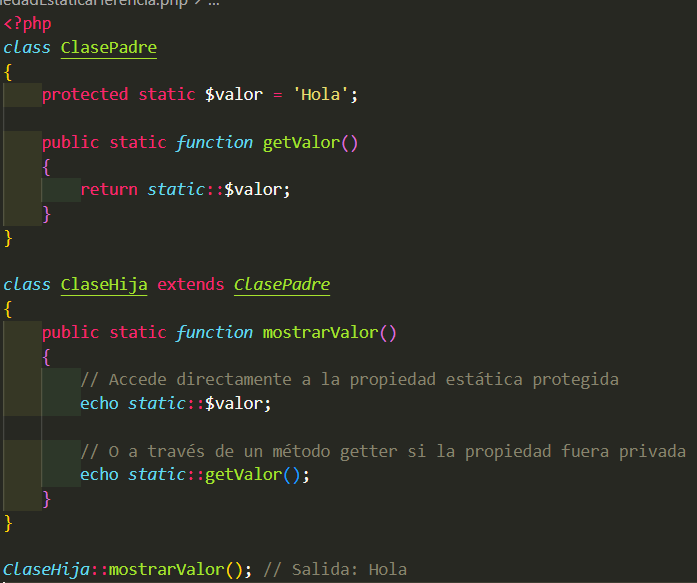


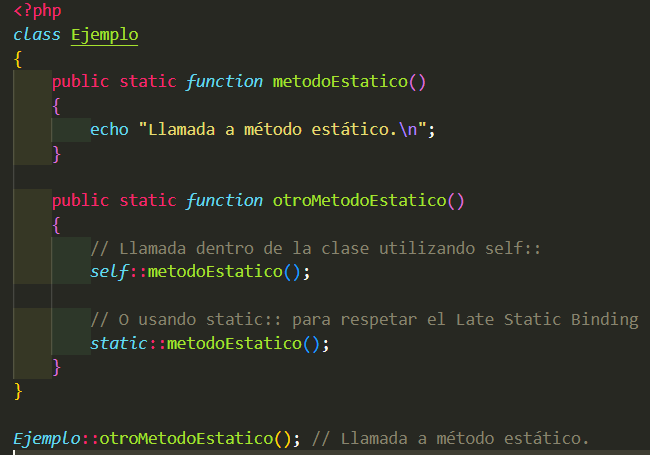
Ilustración 4

En el ejemplo, **“ClaseHija”** accede a la propiedad estática **$valor** de **ClasePadre** utilizando **static::** en lugar de self:: para respetar el principio de Late Static Binding, que asegura que si la propiedad estática se sobrescribe en una subclase, el valor de la subclase es el que se utilizará.

Si la propiedad estática fuera **“private”** ,**”ClaseHija”** no podría acceder a ella directamente y tendría que utilizar un método público o protegido, como **getValor()**, para obtener su valor. Esto es importante para la encapsulación y el ocultamiento de la información, lo cual es una práctica común en la programación orientada a objetos para mantener el código modular y mantenible.

**¿Cómo llamar un método estático dentro de la clase?**

Para llamar a un método estático dentro de la misma clase donde está definido, puedes usar self:: seguido del nombre del método. También puedes usar static:: si quieres que se respete el Late Static Binding (LSB), lo que significa que si el método estático es sobrescrito en una clase hija, PHP llamará al método sobrescrito en lugar del padre.



1. Implemente el siguiente código de la Ilustración 5 , anexe el enlace del repositorio.

|  |
| --- |
|  |

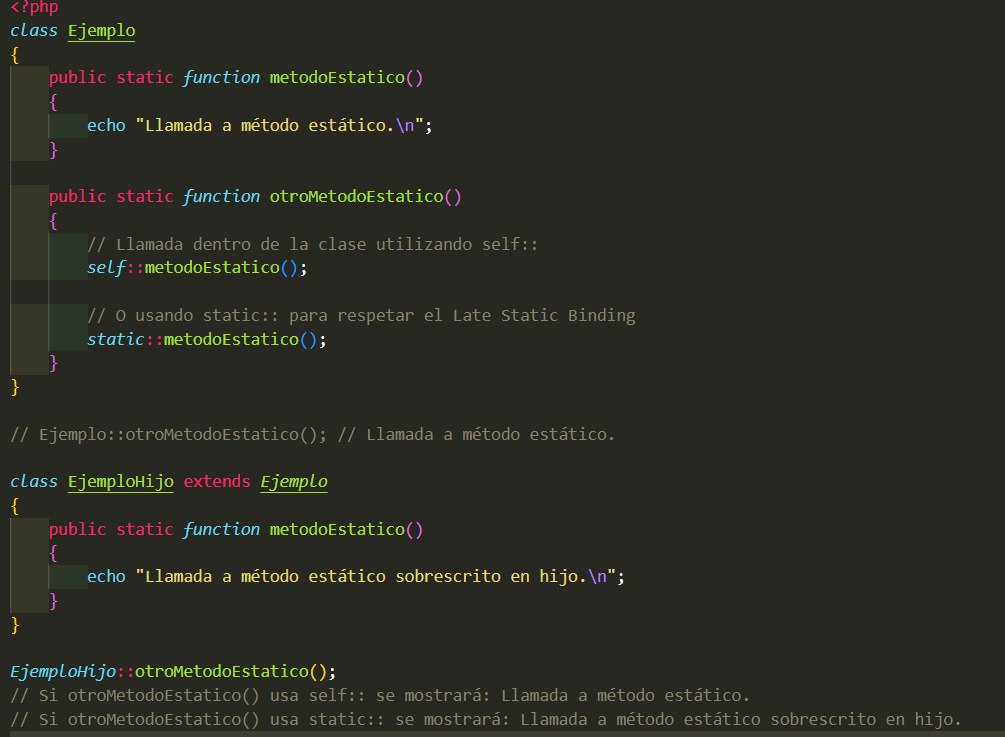
****En siguiente ejemplo, otroMetodoEstatico() llama a metodoEstatico() usando tanto self:: como static::. La diferencia entre self:: y static:: es significativa cuando se utiliza herencia:

Ilustración 5

Si otroMetodoEstatico() usara self::, siempre llamaría al método definido en Ejemplo, incluso cuando se llama desde la clase EjemploHijo. Si otroMetodoEstatico() usara static::, llamaría al método definido en EjemploHijo cuando se llama desde esa clase, porque static:: respeta el LSB.

Usar static:: es generalmente la mejor práctica cuando se llama a métodos estáticos que podrían ser sobrescritos en clases derivadas, porque permite que el polimorfismo funcione con métodos estáticos.

**¿Cómo llamar un método estático dentro de otra clase?**

Para llamar a un método estático desde fuera de la clase donde está definido, usas el nombre de la clase seguido del operador de resolución de ámbito (::) y luego el nombre del método.

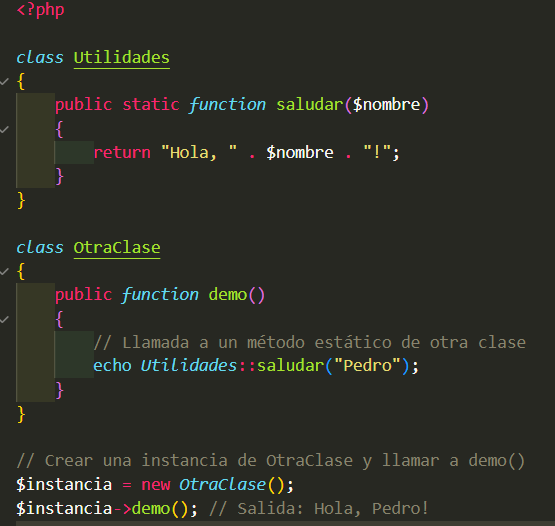
****

Ilustración 6

En este código, “**OtraClase**” llama al método estático saludar de “**Utilidades**” directamente. No se necesita una instancia de Utilidades para llamar a saludar porque es un método estático.

1. Implemente el código de la Ilustración 6 y 7, anexe el enlace del repositorio.

|  |
| --- |
|  |

También puedes llamar a métodos estáticos de otra clase desde dentro de los métodos estáticos de tu clase:

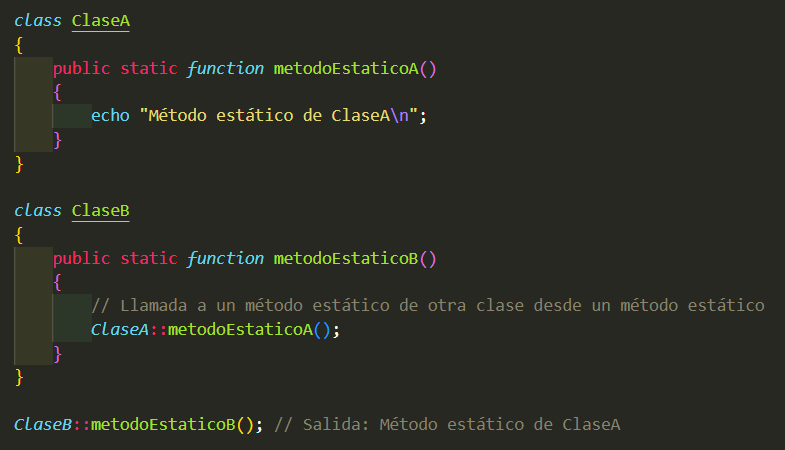


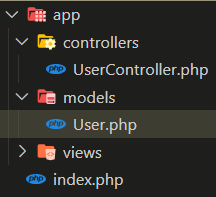
Ilustración 7

**ClaseB::metodoEstaticoB** llama a **ClaseA::metodoEstaticoA.**

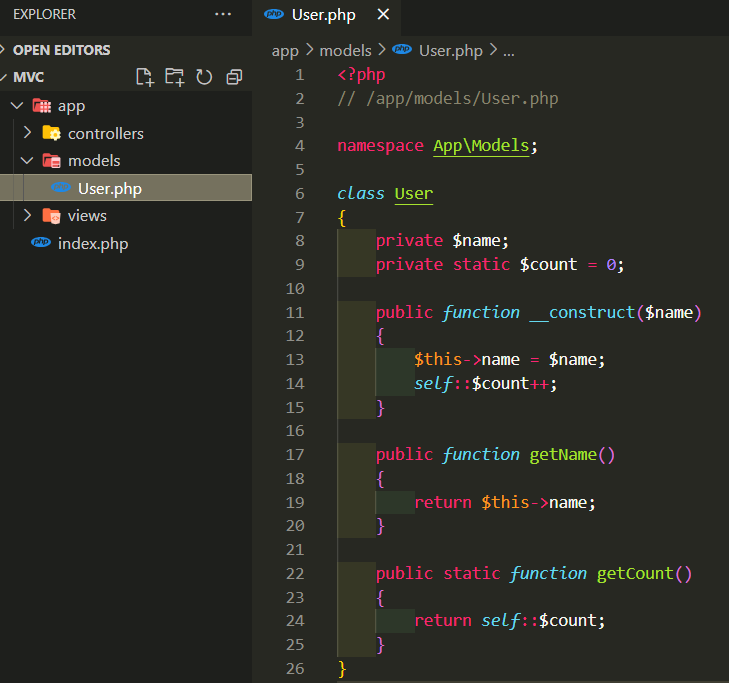
No importa si el método que hace la llamada es estático o no; el acceso a los métodos estáticos de otra clase siempre se realiza mediante el nombre de la clase y el operador de resolución de ámbito.

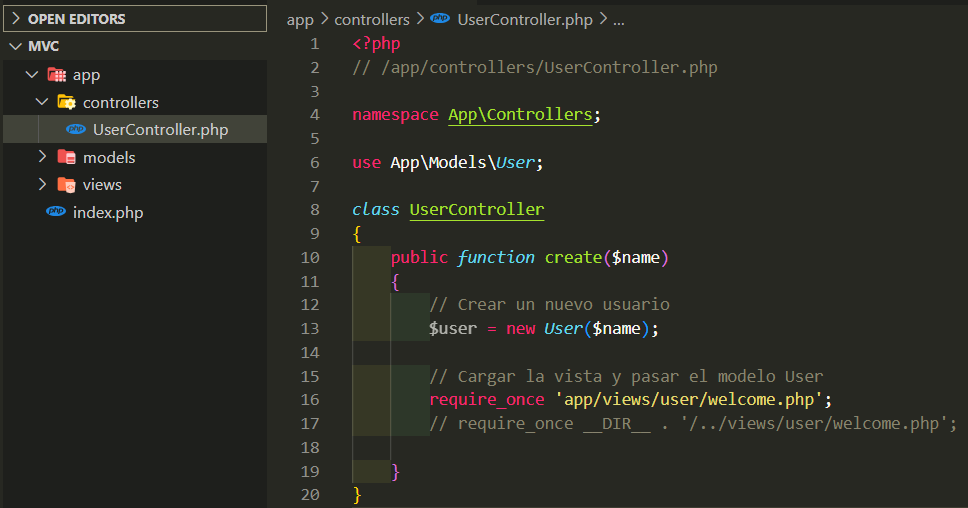
1. Desarrolle el siguiente proyecto y anexe link del repositorio.

|  |
| --- |
|  |

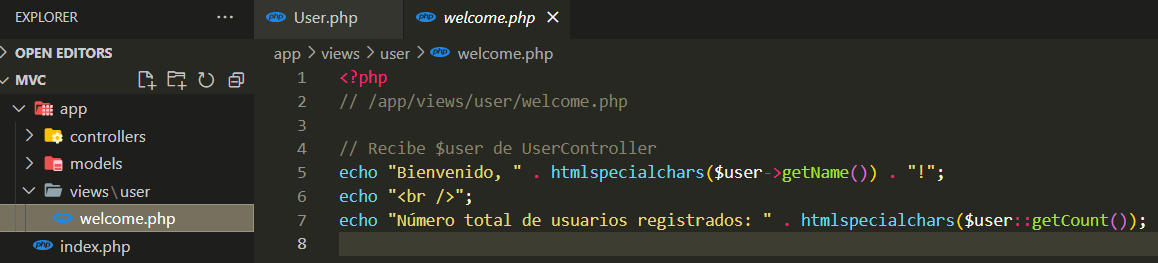
Vamos a construir una pequeña aplicación que gestiona los usuarios de un sitio.

Veremos cómo se definen y se utilizan propiedades y métodos estáticos, cómo se accede a ellos dentro y fuera de las clases, y cómo funciona todo esto dentro de la estructura MVC.

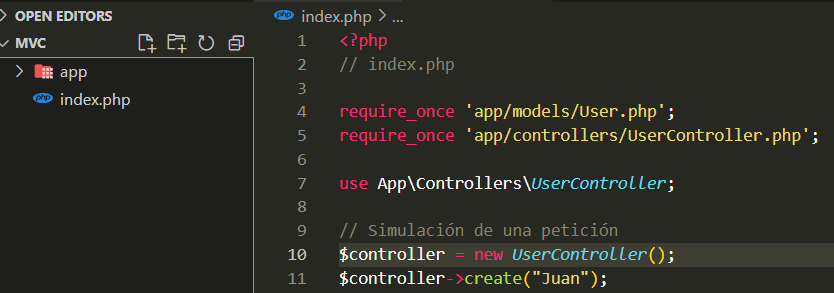
**Modelo (User.php):** El modelo User tendrá un método estático para registrar usuarios. 

**Controlador (UserController.php):** El controlador UserController manejará la creación de usuarios y su visualización.

**Vista (welcome.php):**Una simple vista que muestra un mensaje de bienvenida al usuario.



Un front controller simple para manejar la petición y utilizar el controlador.



El anterior ejemplo muy simplificado y no incluye algunas prácticas y características de seguridad importantes (como la sanitización de entradas, el manejo de errores y la carga automática de clases). También, en una aplicación real se utilizaría una herramienta como **Composer** para la gestión de dependencias y la carga automática de clases, y probablemente se añadirían más capas de abstracción para gestionar las peticiones y las respuestas, así como la interacción con una base de datos.

1. Suba en Git y anexe los proyectos de la guía en un archivo .zip documentos requeridos.

<https://github.com/Monsy017/Propiedades-Metodos-Estaticos.git>

1. Subir la evidencia al LMS